Six Facets of the Open COTS Box

(March 2000)

Daniel H. Dumas
Certified Consultant IT Architect, Network Computing
IBM Belgium
Square Victoria Regina 1
B-1210 Brussels, Belgium

Summary

Although procurement of COTS software for Defence applications has long included evaluation in terms of the products' respect for standards and norms, actual experience has often revealed shortcomings in the ability to deploy solutions based on these packages widely over a period of time. We look here at what additional factors need to be considered in order to make the use of COTS software more likely to bring continuing benefits over the life of an application system. The six aspects that are considered in the paper are:

- Presentation interfaces
- Release compatibility
- Portability
- Programming interfaces
- Security interfaces
- Management interfaces

Introduction

The advantages of using COTS software packages and components are widely known and appreciated, namely:

- Rapid availability (by definition: off the shelf)
- Lower initial costs (because fixed development costs are spread over a wider user population)
- Widespread and higher quality education offerings (again because of the wide user base)

In choosing a particular software package, an organisation will also look to such factors as the ability to use it on the platforms that are most widely used within the organisation (including possibly heterogeneous platforms), and the breadth of applicability of the solution, to understand the economies of scales and of skills that can be realised. This touches upon factors commonly referred to as the "openness" and the perenniality of the solution.

Although procurement of these offerings for Defence applications has long included evaluation of how well they respect official standards and norms, actual experience has often revealed shortcomings in the ability to deploy them widely even over a few years' time. We may wonder, then, what additional factors need to be considered in order to make the use of COTS software more likely to bring continuing benefits over the life of an application system. We will consider here six facets of the definition of "open software" that need to be taken into account in evaluating COTS offerings. These will be presented symbolically by looking at the kind of information we might hope to find written on the six sides of the cardboard box that the COTS software is delivered in.

Six Facets that should be considered in COTS software evaluation

Each of the six sides of the box that a COTS software is typically delivered in can serve to remind us of a separate, important aspect that needs to be considered in evaluating the software, in order best to ensure its long-term applicability in a particular environment.

Paper presented at the RTO IST Symposium on "Commercial Off-the-Shelf Products in Defence Applications "The Ruthless Pursuit of COTS"", held in Brussels, Belgium, 3-5 April 2000, and published in RTO MP-48.

Those six aspects, that we will detail in the rest of this paper, are the following:

- Presentation interfaces
- Release compatibility
- Portability
- Programming interfaces
- Security interfaces
- Management interfaces

Presentation interfaces

Let's consider the top of the box first - and with it, the first thing that you see when you look into a new software: the presentation interfaces.

This aspect , especially when we are dealing with graphical interfaces, has an undeniable emotional impact at the time of product selection. Ergonomy of use in some cases may merit a detailed and objective study, including how long it takes to accomplish some benchmark series of tasks, for both the inexperienced and the experienced user. It remains, of course, difficult to quantify to what degree evaluations are conditioned by more subjective elements such as the use of colour and graphical elements, and the aesthetic elements of information layout.

The nature (and in particular, the complexity) of the presentation interface can also have an impact on performance in environments with bandwidth or processor constraints. It is appropriate that this be taken into consideration at the time of evaluation, but this aspect is rarely apparent in a demo or pilot environment, where bandwith and processor capacity is typically unconstrained.

But beyond considerations such as usability and attractiveness, the choice of presentation interfaces has a real impact on the ability for a product to be used at various locations within the organisation, and to integrate with various other applications.

If a single type of user interface is used, an approach that presents significant advantages is to communicate to the user interface with a Web Browser-supported data stream, such as HTML, XML or client-side Java functionality. The Web Browser, of course, has the distinct advantage of running on multiple platforms.. It can also connect to multiple servers simultaneously. This allows easy passive integration, as well as active integration via hyperlinks. Increasingly, as well, it can be used to provide slightly different presentation interfaces to users according to their individual preferences, through the use of style sheets.

An approach that goes further than this is to support multiple user interfaces. Indeed, if sufficient consideration is given, at the time of application development, to the separation between presentation functions and business logic, the same application can be designed to work indifferently with various interfaces. As possible interfaces, we might imagine the following:

- A non-graphical, or transactional, interface to the server functions, via a specific Application Programming Interface or a Messaging interface
- A specific Client GUI (graphical user interface)
- A standard Web Browser GUI
- An interface to portable devices such as Personal Data Assistants, cell phones or pagers
- An output interface such as print, e-mail, pager or FAX
- A telephony user interface, which could be touch-tone or voice activated

There are multiple approaches possible to providing universal presentation interfaces. The intelligence required in order to adapt the presentation format to a particular device can either be located in an intermediate server, or *transcoder,* or can be built into the device itself, based on standard datastreams that are defined to be applicable for data transmissions to a wide variety of devices. It is important to see which of these approaches is adopted by a particular software package, and to evaluate how the approach fits in with the planned deployment of various user interfaces within the organisation.

The idea of universal access to applications is gaining credibility through recent advances in the standardisation of separation between content description and layout, based on the use of the eXtensible Markup Language (XML - which, in contrast to HTML, is not a markup language in the sense of presentation layout, but a content description language) and the eXtensible Style Language (XSL) specifications.

Both the transcoding and the device-resident style sheet approach can provide increased flexibility for customisation of the presentation interface. Typically, though, they will be used for different types of applications.

The approach that will leave the greatest flexibility for customisation of the look of the presentation interface by individuals will be the approach based on a style sheet selected at the device. This will generally allow the user to adjust the presentation interface without requiring any modification to the business logic. It's an approach that has advantages for the editor of the software as well: it allows them to avoid developing specific customisation Application Programming Interfaces (API's), and will reduce needs to provide access to source code, with the accompanying negative impacts that has on maintainability of code and on the ability to protect intellectual capital and software assets.

There are other types of applications where the transcoder approach is more powerful, of course: in cases where the datastreams are non-standard or unstructured, for instance. One application where use of this kind of service has appeared recently is in providing multilingual interfaces to a single-language application. The transcoder in this case is used to accomplish translation of text on-the-fly.

Release compatibility

Let's go back to the imaginary box that our software has just been delivered in, and take a look at the front side now. What kind of things might we find there? Version X.Y.Z. New! Improved! Bonus!: now includes product ABC (demo version).

Questions we might ask upon seeing all this include:

- Why the new release? (as well as: when was the previous release? when is the next one planned?)
- What statement of requirements motivated the new function and improvements?
- How does the additional product included in the package affect me and what dependencies does it create?

COTS software is fundamentally oriented to a mass market. New releases can serve a number of purposes in this environment:

- They are a delivery mechanism for error maintenance
- They are a mechanism to generate renewed interest
- They incorporate new technology
- They allow adjustments in positioning relative to competitive products, and to products the vendor owns, has acquired, or is forming marketing alliances with

Because market share is an important consideration, COTS products typically try to cover as large a spectrum of function as possible - sometimes earning them the reputation of "bloatware"! This inevitably results in their containing features that are not strictly required for a particular application. Additionally, and more importantly, the focus very often turns to rapid product cycles rather than to managed change.

In this context, it is not infrequent to see problem determination and the application of fixes reduced to very minimalist proportions: install the next release and hope your problem goes away. The policy of maintenance for releases for a particular COTS product certainly merits investigation. Is corrective maintenance available between releases? How can it be delivered?

Release "churn" has a negative influence on the length of availability and on the effective support period of a release. How frequently do releases change? Is this acceptable in terms of the period foreseen to roll out a product to the various users in the organisation?

Change management considerations apply not only to the area of corrective maintenance, but to the implementation of functions that have been requested as part of the product requirements process. Historically, input to the requirements process has been restricted to a small number of influential players. The advent of the Internet is changing that in certain IBM and Lotus laboratories, where requirements from a much broader public of developers is solicited during the product development cycle.

Eventually, even if hopefully not during the initial rollout period, the organisation will probably end up considering migration of users from one release to another, or one version to another, of any given COTS product. A number of other important questions will inevitably arise at that time: will the things that I have customised or developed continue to work with the new release (forward compatibility)? Will the things that I was doing with the previous release continue to work with both the new and the old release if I perform them with the new release (migration compatibility)? Will the fact of using the new functions in the new release prevent me from interoperating with users using the old release (backward compatibility)? Is there some way to configure so that compatibility is ensured (e.g., disabling the use of the new functions or new data formats until migration is complete)? What needs to be done to move users from the old release to the new release, and data from the old release to the new release (migration planning)?

Although forward compatibility is widely practised, and backward compatibility is sometimes possible, the ability to configure for automatic backward compatibility is rarely foreseen. These last considerations can however be especially important in situations where upgrade decisions are taken by independent entities or distributed entities that need to interoperate.

Obviously, these are all essential change management considerations that need to be understood before a given organisation leaps into a new release. But when we consider the question of release compatibility across organisations, the question gains a new dimension of complexity: Organisations throughout the world are not marching in lockstep. Different organisations are doing different things at different times. No version plan could ever be made that would make all of a COTS supplier's customers happy.

Customers have to have the discipline to navigate through releases, and have some restraint to do version control. Industry on the other hand, who too rarely make public commitments to maintenance of a particular release, could do better to provide maintenance of prior versions over a fixed minimum number of years. But this does not appear to be a prevalent trend. I do have one COTS software box that states: "Maintenance will be provided until .... No maintenance will be provided after that date". It just happens to be for IBM DOS 4.0!

New releases can also entail additional licensing fees in addition to the unaccounted human costs associated with the installation, configuration and problem determination efforts required for those new releases.

Portability

Moving on to the right side of the box, we might see a text such as the following: "Requirements: Windows 95 or 98, Intel Pentium 133MHZ or greater with a minimum of 24MB, a Sound Blaster compatible sound card and SVGA graphics capability configured for at least 800X600 resolution." A number of questions might typically come to mind:

- Will this software be applicable to my other machines that work with other hardware

and/or other operating systems? Or will I at least be able find the equivalent software available for the other operating systems?
- Will this work with the new operating system version (NT 4.0, Windows 2000, etc.) that I am planning on installing (or that I will be forced to install for some other reason)?
- Will it interoperate with my other systems, or with the systems in other organisations that I need to deal with?
- How scalable is the package? Can it take advantage of additional memory, additional processors, additional machines or more powerful machines, in order to accomplish more work?

Portability has to do with flexibility across technologies and over time. Whereas in the considerations concerning release compatibility we were considering constant platforms and varying software, here we are considering constant software and varying platforms.

We are looking to be able to deploy a COTS-based solution widely and to keep it viable over a number of years, eventually in a number of different organisations that need to work together. In order to accomplish that , we need to accommodate changes in technology and possible changes in hardware and operating system vendor strategy. The rhythm of change of hardware and operating system technology continues "relentlessly" as well! It is therefore desirable for the software to have a high level of abstraction from the hardware and operating system level.

Packages and components that we can characterise in this way generally are designed to run on multiple platforms today. The greater the number of platforms supported, the greater the effective openness of the software.

A major advance in portability has occurred recently with the advent of the Java Virtual Machine (JVM) and the standards that have been defined in the area of application development based on the Java language. The JVM, implemented across various hardware and operating system platforms, allows the same "100% Java" byte codes to be executed in the same way regardless of the instruction set and services of the underlying physical machine and operating system. The principle is: "Write once, run everywhere". Though there is a certain overhead associated with this additional level of abstraction, techniques such as Java compilers and Just-in-time Java compilation now allow performance-critical processes to achieve results that reasonably approach the performance of native instruction-set execution for equivalent functions. Java-based applications that correspond to user performance expectations are increasingly becoming available, and this trend can be expected to continue

Portability is also affected by the architectural approach followed by the solution. Functions that risk being dependent on specific platforms can be separated from the other functions, and accessed via a protocol that allows the function to be located elsewhere, in order to make the overall functions accessible from a wider range of platforms. This is essentially acknowledging that portability is most important across the machines that have the greatest number of instances installed, while an organisation can afford to have a limited number of servers for which portability is not considered an issue (typically, which have specific characteristics of availability, performance, security, or other criteria).

In terms of the communications protocols used between the dissociated functional layers, we can speak of synchronous protocols (requiring both sender and receiver to be active and accessible - e.g. as in the use of a TCP socket-to-socket protocol) and asynchronous protocols (allowing processing to go on with guaranteed delivery at a later time, when the receiving application is not active or accessible, e.g. as in MQSeries message queuing). Both of these types of communication are available on a wide variety of platforms.

The various approaches to separation of functional elements can be characterised in terms of architectural tiers. Although there are different approaches to counting the number of tiers involved, the following should be generally recognisable to everyone, at least in theoretical terms. It is presented here in an order that corresponds in general to an increasing order of portability:

- Monolithic applications
- Two-tier client/server applications (presentation function located in the client, communicating to business logic in an application server with integrated data store)
- Three-tier client/server applications (presentation function located in the client, communicating to business logic in an application server, communicating in turn to a data store server)
- Four-tier client/server applications (simplified presentation function located in the client ("light client" = browser), some presentation logic located in the web server, business logic in an application server, communicating in turn to a data store server)

Separation of the function into additional tiers increases their independence. It makes it easier to "live with", and integrate to, those isolated elements in the overall solution that are the most difficult to change and that may have the greatest need for stability and the least portability.

Programming interfaces

On the back of the box, we often find some information about the interfaces supported - though perhaps not nearly in the detail needed in order to put them into practice! In order to extend a package or integrate it into a larger context, programming is often necessary. How easy - or difficult- will that be with the package at hand?

Often this is a question of experience, and Internet sites for developers and user forums can provide interesting insights sometimes. But there are certain interfaces that provide very high-level function, and therefore can be used very productively. An example of such high-level function is that provided by such a specification as Enterprise Java Beans (EJB). In addition to the support for the Java language and Object Request Broker for connection to the function of other (eventually remote) objects, the EJB container provides transactional functions (such as management of the unit of work and scope of recovery), session management functions (via EJB Session Beans), persistent data store functions (via EJB Entity Beans) and access control functions.

We can distinguish multiple levels of openness in the area of programming interfaces. We can find:

- Undocumented/unofficial interfaces (true "proprietary interfaces")
- Official interfaces with limited programming function (e.g. "wizards")
- Official interfaces in a proprietary programming language (e.g. Oracle PL/SQL script)
- Official interfaces in a non-proprietary programming language (e.g. the use of COBOL, C or Java)

The usefulness of the limited-function interfaces is also conditioned by which middleware they foresee. Sometimes a small door can open onto a very large playing field! Take, for instance, a communications interface to SMTP, to EDI, or to MQSeries. Or take an SQL API, allowing the relational database to serve as an integration point to other processes, which might run asynchronously, or synchronously through triggers or through stored procedures.

The web browser, with its capability of being in fact a client to multiple servers at the same time, even on the same web page, and with its programming capability, also provides an integration point for applications, provided of course that the application foresees using a web browser interface.

For server-type implementations, additional technical analysis of the limitations of the interfaces can also be important. Such items as

their support of caching, buffer handling, threading and connection pooling can have an important impact on their scalability..

Security interfaces

On the left side of the box, it would be nice to see something about how the package handles security and access control. Does it provide and use its own system? Does it build on the facilities of the operating system?

Even the most mundane applications (for instance - a word processor!) may have to handle personal, restricted or confidential information. Issues such as user identification, authentication, access control, encryption and non-repudiation must be addressed. At times, the operating system can be counted on to deliver these functions (when, for instance, it is a question of providing access control for information located on the machine). At other times, certain aspects need to be handled on an application level (for instance for data that needs to be transported, that needs to be digitally signed, that needs additional granularity of access within a given file, etc.).

The interfaces available within a COTS package can determine whether these aspects will require (or even allow) specific development, whether it will work with existing infrastructure (such as smart cards, readers, digital certificates, directories, existing definitions of users, groups and roles, encryption algorithms, etc.) or whether a separate infrastructure will need to be set up and maintained.

One approach taken in this area is that of providing a standardised interface to an external "pluggable security module", which can provide cryptographic services of various sorts. This is the CDSA model, originated by Intel, and used in various recent IBM products today. It is also the model being used by Lotus to separate the grade of security provided in a particular environment from the actual standard function of the underlying messaging product.

Management interfaces

And now for the side that everyone forgets to look at. Until there are problems, that is!

Typically we are going to be rolling out COTS software to large numbers of users, perhaps in various distributed locations, and then we are going to have to maintain an inventory of who has what level, detect problems that might occur, manage the application of maintenance, perform backups, provide for recovery, maybe provide remote debugging or remote assistance, operate; monitor performance and availability, etc..

How do we manage the cost of doing that?

The Management and Monitoring Interfaces provided by software applications can in theory support multiple objectives in the organisation, including failure detection, deployment tracking for initial roll-out or maintenance, detection of misuse, assembling and tracking performance data, remote operations, assistance or debugging, etc. But there must also be some coherence in the interfaces provided across the various applications in order for this to be viable.

COTS software will generally lack these capabilities to manage themselves. What is more important is that they interface to some central management and monitoring facility that does have these capabilities. Here is an area where the Programming APIs can come to the rescue. They can allow alerts to be implemented relatively easily, based on some reusable standard functions. An example of this are the Java classes (functions) provided to send alerts to the Tivoli Enterprise Management facilities.

Conclusion

By considering these various facets of openness, IT architects can improve the use of COTS components in complex Information Technology projects. It took some time for software

companies to embrace the open movement. Today, with companies increasingly responsive to customer needs, and new technologies addressing a broader range of interfaces, we are ready to move forward to a more comprehensive definition of openness, and, as shown by a few of the examples from IBM related in this paper, we can expect that the companies providing Commercial Off-The-Shelf software will be prepared to respond.